TMSTAF – The Extended Use of STAF on Test Automation

Bevis Lee
Trend Micro Incorporation, Taipei, Taiwan
bevis_lee@trend.com.tw

Abstract
As software packages becomes increasingly large and complex, the time required for testing them, throughout the development lifecycle, has also increased. Because testing activities consume the majority of software Quality Assurance (QA) resources, a test platform was needed to speed up test cycles without any decrease in test result accuracy. The use of Python 2.6 scripting language to create a faster, automated testing platform is reported here. Trend Micro Software Testing Automation Framework (TMSTAF) was developed using Python 2.6, based on Software Testing Automation Framework (STAF), to improve automated testing. We found that TMSTAF not only decreased testing time, it provided faster process integration, test feedback, and improved overall software quality. Using the TMSTAF automation environment for development and execution is simple to set up. Test cases can be created for use in both manual and automated tasks, converted to automated scripts, and implemented with structural and flexible mechanisms. Automation script pre-runs and debugging make troubleshooting more efficiently; TMSTAF test report data can be used to identify quality issues quickly. TMSTAF can be seamlessly integrated into the build release process, making it a smart option for software QA engineers.

Keywords: *TMSTAF, test automation framework, automated software testing.*

1. Introduction
As software becomes increasingly large and complicated, QA teams must handle hundreds or even thousands of test cases. The time requirements for QA testing consume the majority of the product development lifecycle because it can take several weeks to verify a single function within the product. Due to human resource constraints and the time that other items besides function verification require (system integration), stress tests are decreased or even waived. Automating tests is the most efficient way to manage project teams. However, it is important to figure out how to find a test-automation framework conforming to product characteristics. TMSTAF provides functions for process integration, cross-platform and result gathering, and feedback testing. With TMSTAF, QA can automate testing and integrate with the software development process more easily. Leveraging TMSTAF helps to improve the software quality and speed of software development.

2. Framework and Features
TMSTAF was developed by Trend Micro Inc. in 2007, using the Python 2.6 scripting language. It supports Windows, Linux and Mac OS, and uses STAF (http://staf.sourceforge.net) as the base of the cross platform test.

2.1. TMSTAF Framework

As shown in Figure 2.1, TMSTAF framework can be divided into three layers:

- Kernel modules: Composed of Python and STAF components, it handles basement and cross-platform operations, etc.

- TMSTAF/Common modules: Composed of TMSTAF basic modules, cross-platform process management modules and common modules. It handles test script execution, test result consolidation and cross-platform test operations updated with all major functionalities. Common modules include file system save/read, Windows registry save/read.

- Products customized modules: Modules and test scripts can be developed according to product characteristics, and the test execution process can be customized as well.

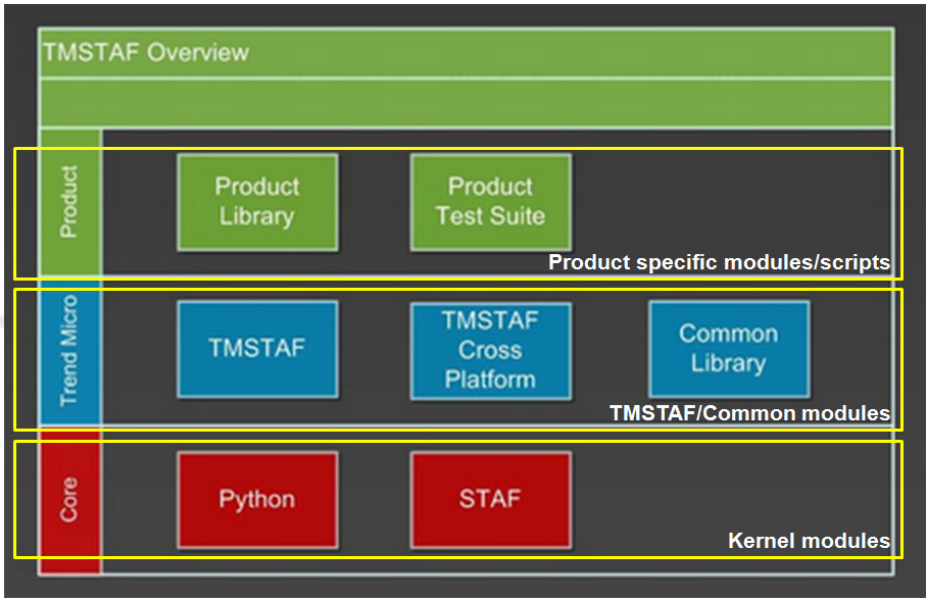

Figure 2.1. TMSTAF Framework

The three-category file structure of TMSTAF is inherited from STAF (see Figure 2.2):

- Module files: Store common or self developed Python module files.

- Test suites/test case files: Store product specific test cases and related files, such as virus log estimation or setting files.

- Test results and execution record files: Store test results and completed execution logs.

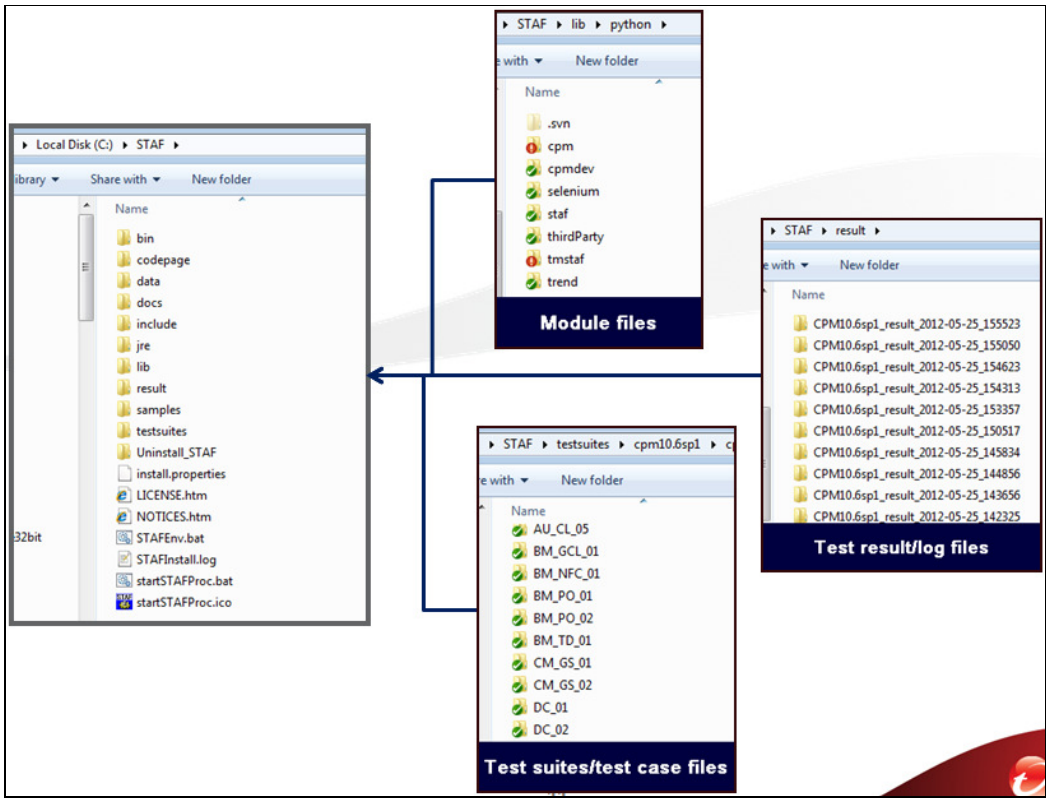

Figure 2.2. TMSTAF File Structure

2.2. TMSTAF Features

This section describes the six primary functions of TMSTAF:

- Introduction to testware concepts: The product can be treated as testware and users can modify settings such as test execution according to product characteristics. It simplifies development by defining test suite and test cases with Python grammar.

- Introduction to the test process: Integrate the test case executing process with the framework according to actual situations. Process includes:

    - Setup: Test enviorment initialization, includes any preparations before test.

```
objFunction=PythonFunction('trend.fileUtil','removeFile')
result=self.addSetup(objFunction,[destSystemRoot+r'\tsctestr.exe'])
```

    - Test: The steps of test execution

```
objFunction=PythonFunction('until.manual_scan.manual_scan_utility','startM
anualScan')
result=self.addTest(objFunction,[ppManualScan_Proc_Port,'QA_SCAN_MODE_QUIC
K'])
```

    - Teardown: Restore actions after finishing test

```
objFunction=PythonFunction('until.dbUtil','clean_db')
result=self.addTearDown(objFunction,[DBFile,''])
```

- Support command line interface (CLI): Figure 2.3 shows how TMSTAF is executed in terms of CLI, which is easily integrated with the continueous integration or batch system.

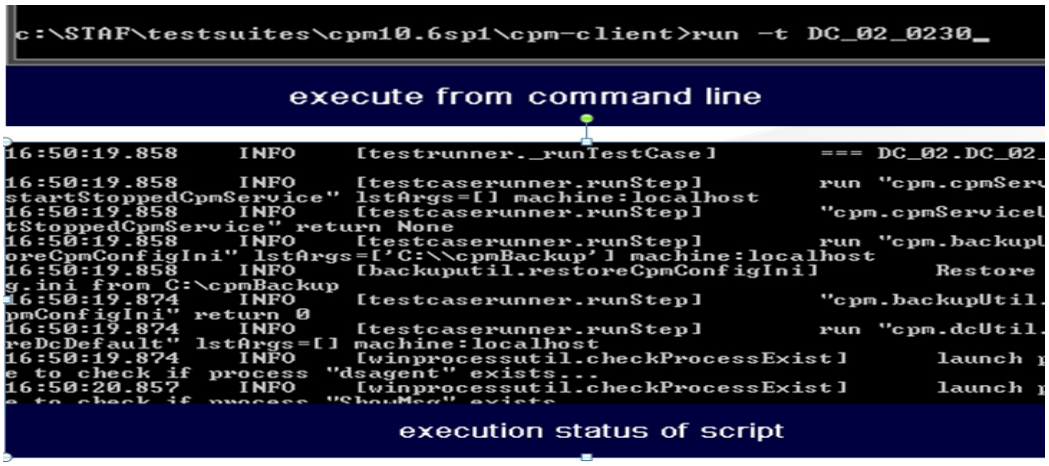

Figure 2.3. Execute Test by CLI

- Execution log: Figure 2.4 shows how the execution log saves the execution process as a file to simplify debugging and issue tracking.

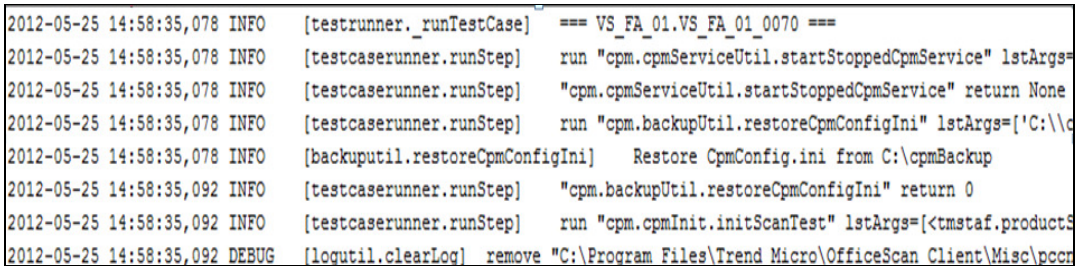

Figure 2.4. Execution Log

- Figures 2.5 and 2.6 show the complete test report and email notification, respectively. They contain enough information to quickly understand the quality of product.

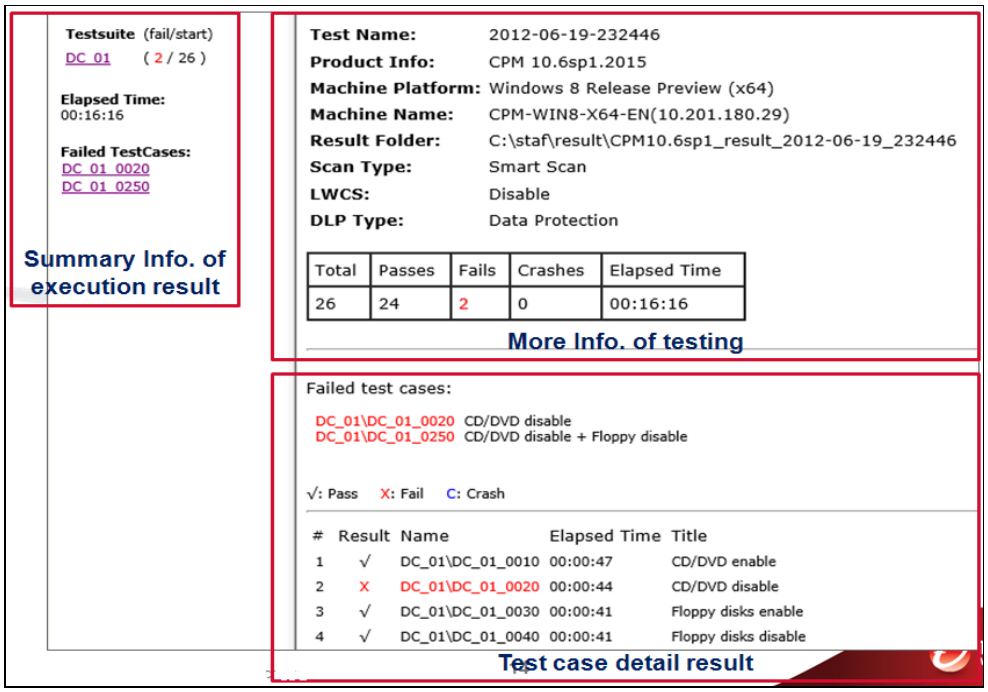

Figure 2.5. Test Report

Figure 2.6. Email Notification of Test Result

- Cross-platform testing: Enable cross-platform testing by STAF and TMSTAF modules, as shown in Figure 2.7, the test can be executed by the client assigned by the central automation server.

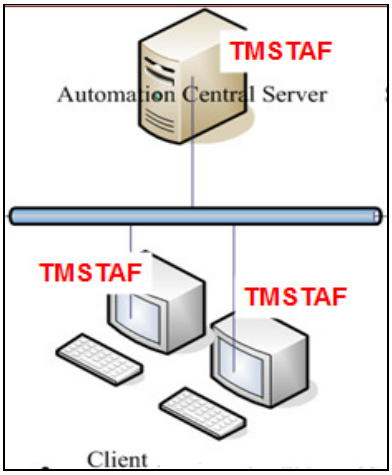

Figure 2.7. Cross Platform Test Diagram

2.3. Major Differences between TMSTAF and STAF

As shown in Table 2.1, TMSTAF provides more test automation features than STAF.

Functions	TMSTAF	STAF
Management of test suites and cases	Supported	Not supported
Management of test process	Supported	Not supported
Executing tests from a CLI	Supported	Not supported
Customized test execution logs and reports	Supported	Not supported
Management of cross-platform testing	Supported	Not supported[1]
Programming language support	Python 2.x only	Multiple languages

Table 2.1. Functonal Differences between TMSTAF and STAF

3. TMSTAF and Test Automation Development Process

This section describes the test automation development process and the types of support that TMSTAF provides to developers. As shown in Figure 3.1, the test automation development process can be divided into two phases.

[1] Need to combine with STAX

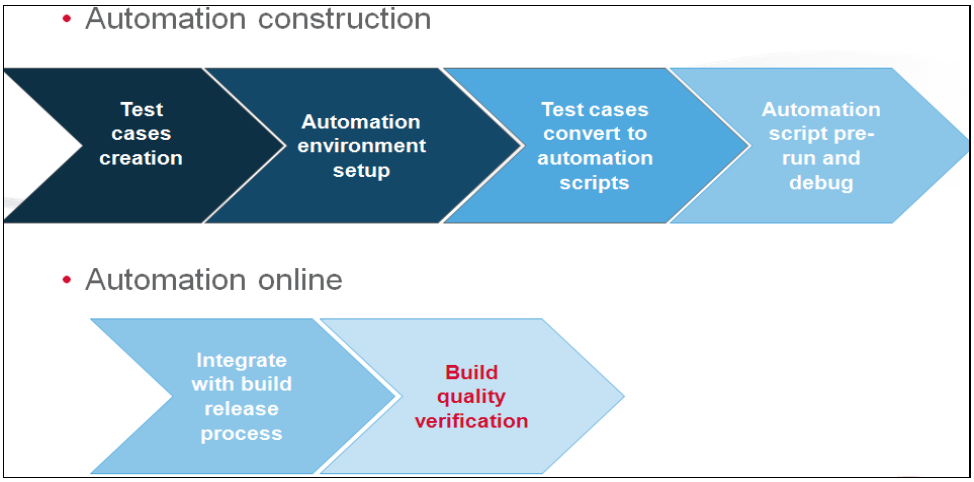

Figure 3.1. Test Automation Development Process

- Build up the test automation:
  - Test cases creation: As shown in Figure 3.2, the correlated test cases TMSTAF can be matched with those on test case management system (such as SCTM, http://www.borland.com/products/SilkCentral/) instead of creating new cases specified for automation test.

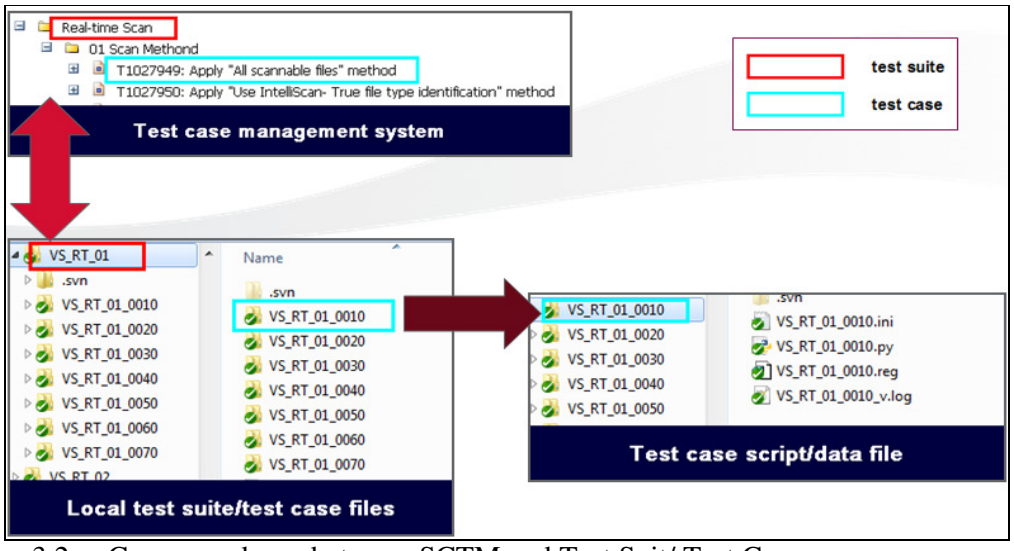

Figure 3.2: Correspondence between SCTM and Test Suit/ Test Case

  - Test environment installation: As shown in Figure 3.3, the environment can be set up quickly by installing three components, configuring basic settings.

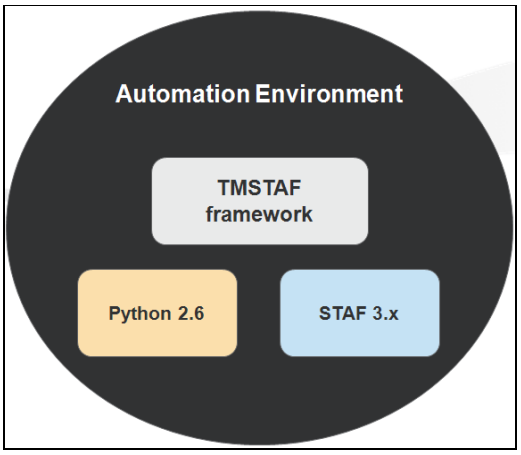

Figure 3.3. Development and Test Enviroment

- Convert the test cases into the automation script: A structured method (automation modules → test steps → test script) can be used to transfer test case to script step by step (See Figure 3.4.).

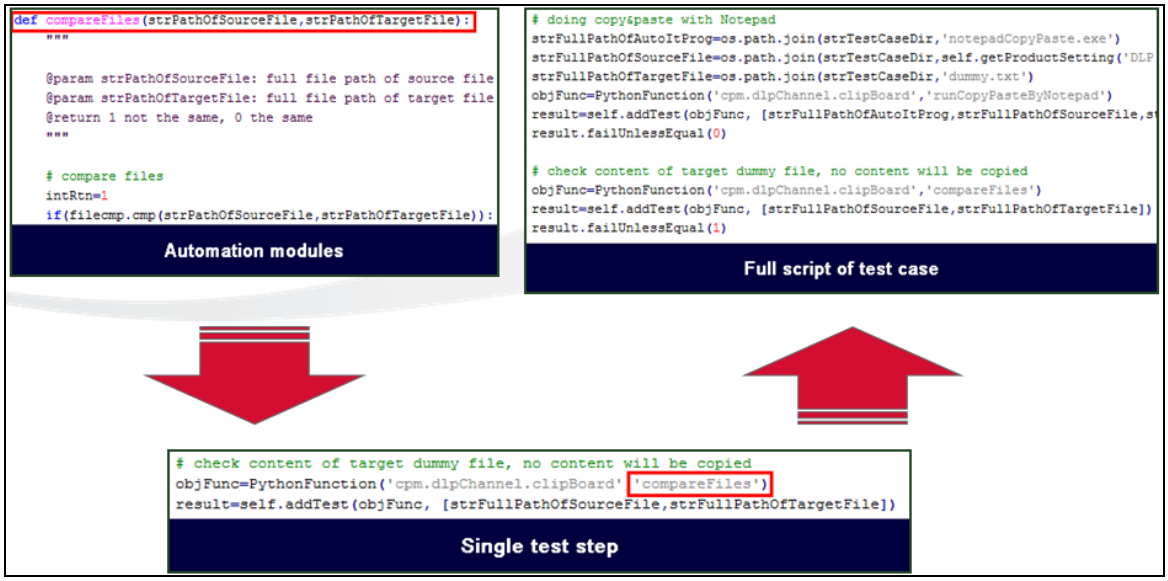

Figure 3.4. Transfer Test Case Structurally

- Automation script debugging: As shown in Figure 3.5, the execution log provided by TMSTAF has enough information for efficient debugging.
  - Inherited from python, three levels of log information are provided, INFO, DEBUG and ERROR
  - Execution result of automation module, ex. testcaserunner.runStep

```
2012-05-24 14:50:03,171 INFO    [testrunner._runTestCase]   === WR_01.WR_01_0020 ===
2012-05-24 14:50:03,171 INFO    [testcaserunner.runStep]    run "cpm.cpmServiceUtil.startStoppedCpmService" lstArgs=[]
2012-05-24 14:50:03,171 INFO    [testcaserunner.runStep]    "cpm.cpmServiceUtil.startStoppedCpmService" return None
2012-05-24 14:50:03,171 INFO    [testcaserunner.runStep]    run "cpm.backupUtil.restoreCpmConfigIni" lstArgs=['C:\\cpm
2012-05-24 14:50:03,171 INFO    [backuputil.restoreCpmConfigIni]    Restore CpmConfig.ini from C:\cpmBackup
2012-05-24 14:50:03,187 INFO    [testcaserunner.runStep]    "cpm.backupUtil.restoreCpmConfigIni" return 0
2012-05-24 14:50:03,187 INFO    [testcaserunner.runStep]    run "cpm.wtpUtil.initWtpTest" lstArgs=[<tmstaf.productSett
2012-05-24 14:50:03,187 INFO    [wtputil.initWtpTest]   sync local time...
2012-05-24 14:50:03,187 DEBUG   [ntpclient.sync]    Sync system time with 10.201.180.101

2012-07-06 12:38:26,246 ERROR   [cpmsetup.uninstallCpmAndReboot]    Run "C:\TMCPMHelp.dll" failed!
2012-07-06 12:38:26,246 ERROR   [testcaserunner.runStep]    run cpm.cpmSetup.uninstallCpmAndReboot
```

Figure 3.5. Detail Execution Log

TestCase : EV_CL_01_0010

Suite	EV_CL_01
Title	EVT_CPM_CLI_ACCESS
DataFile	C:\staf\testsuites\cpm11.0\cpm-client\EV_CL_01\EV_CL_01_0010\EV_CL_01_0010.py
Module	Window Event Log >> Client >> 01 event log
Scenario	C:\staf\testsuites\cpm11.0\cpm-client\EV_CL_01_scanMethod.py
OutputFile	EV_CL_01_0010.log

Setup Steps: Pass: ✔ Fail: ✘ Crash: ⚠

#	Result	Type	Executable	Args	Expected-Return	Actual-Return	Time
1	✔		cpm.cpmServiceUtil.startStoppedCpmService	[]	N/A	None	0.0s
2	✔		cpm.backupUtil.restoreCpmConfigIni	['C:\\cpmBackup']	0	0	0.0s
3	✔		cpm.cpmInit.initScanTest	[<tmstaf.productSetting.ProductSetting object at 0x01BA7650>]	0	0	1.9s

Test Steps: Pass: ✔ Fail: ✘ Crash: ⚠

#	Result	Type	Executable	Args	Expected-Return	Actual-Return	Time
1	✔		trend.winEventLog.clearEventLog	['application']	0	0	0.1s
2	✔		cpm.callCpmCli	['CONFIG', ['-i', 'C:\\staf\\testsuites\\cpm11.0\\cpm-client\\EV_CL_01\\EV_CL_01_0010 \\EV_CL_01_0010.ini']]	0	0	7.7s
3	✔		cpm.cpmRegUtil.verifyRegistry	['C:\\staf\\testsuites\\cpm11.0\\cpm-client\\EV_CL_01 \\EV_CL_01_0010\\EV_CL_01_0010.reg', 60]	0	0	0.0s
4	✘		trend.zipUtil.unzipFile	['y:\\all_vir.zip', 'E:\\tmp', 'virusvirus']	0	2	0.0s

Figure 3.6. Detail Test Result

- Test automation goes online
    - Software development process integration: TMSTAF can be integrated with SCTM and continuous integration system by CLI.

- Software quality verification: Test report in detail can help QA more clearly locate bugs and provide feedback to developers for prompt fixing. As shown in Figure 3.6, the failed test step is highlighted in red colour.

4. Conclusion

In summary, TMSTAF benefits QA automation in several ways. Test cases can be created for use in both manual and automated tasks with no extra effort required. The automation environment is easy to set up for development and execution. Test cases convert to automated scripts and can be implemented in test cases with structural and flexible mechanisms. Automation script pre-run and debugs are an efficiency method for troubleshooting QA automation. TMSTAF integrates with the build release process. Finally, for future build quality verification, information in testing reports is useful to identify quality issues more quickly.

References

Edward Kit, 1995. Software Testing In The Real World: Improving The Process. : Addison-Wesley Professional

Dorothy Graham et al., 1999. Software Test Automation. : Addison-Wesley Professional

Cem Kaner et al., 1999. Testing Computer Software, 2nd Edition. : Wiley

Brett D. McLaughlin et al., 2006. Head First Object-Oriented Analysis and Design. : O'Reilly Media

Wikipedia, 2006. Test automation framework. Available at: <http://en.wikipedia.org/wiki/Test_automation_framework>

Sourcefourge, 2006. Software Testing Automation Framework (STAF). Available at: <http://staf.sourceforge.net>

Alan Page et al., 2008. How we test software at Microsoft. Microsoft Press.